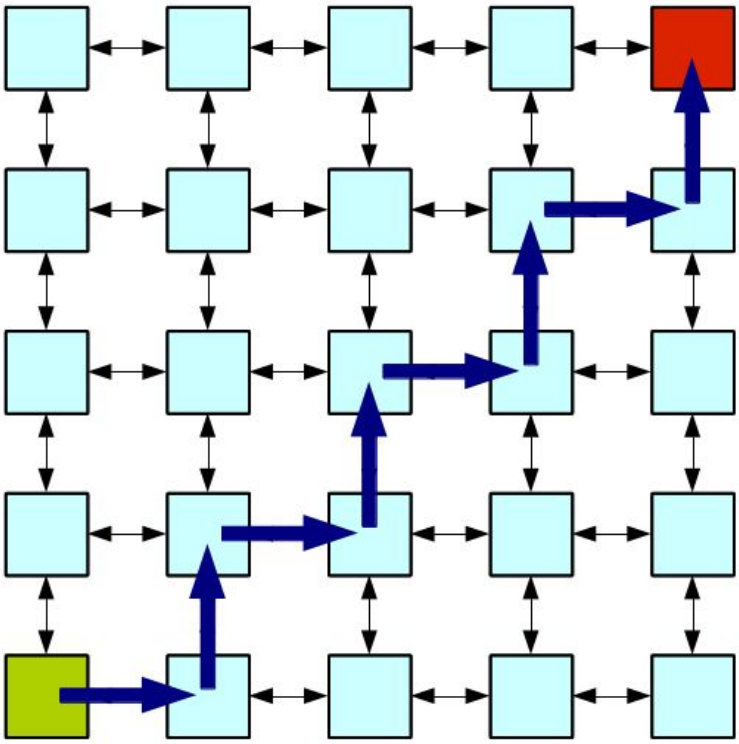
Sistemas de Inteligencia Artificial

Trabajo Práctico:
Algoritmos de Búsqueda



Integrantes:

Juan Franco Caracciolo - 56382

Facundo González Fernández - 55746

Julian Nicastro - 55291

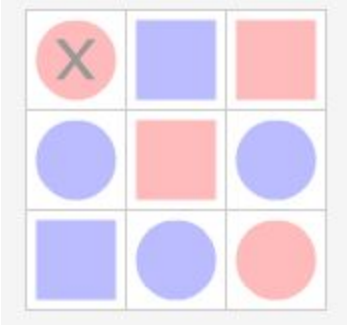
Sebastian Ezequiel Guido - 54432

Objetivos del trabajo

El trabajo práctico consiste en la realización de un motor de búsqueda genérico que pueda resolver cualquier búsqueda de camino informada y desinformada. Además se nos asignó un problema con el cual probar el motor, y para el cual habría que generar heurísticas. En particular el problema de nuestro grupo es el juego “Chain Reaction”

Chain Reaction

Presentacion



El juego consiste de un tablero de NxN celdas donde cada una puede contener una pieza con cierta forma y color, o estar vacía. El objetivo del juego es pasar una vez por cada pieza sin repetir y sin que falte ninguna. Para pasar de una pieza a otra estas deben cumplir 2 requisitos. Las dos piezas deben estar en la misma fila o columna y las 2 piezas deben compartir la misma forma o color (o ambas). Una vez que se haya pasado una vez por cada una de las piezas el juego termina, sin embargo, si se llega a una pieza de la cual no hay más movimientos válidos posibles, se pierde el juego.

Implementacion

Es necesario mencionar algunos aspectos del algoritmo de generador de caminos para tener en cuenta a continuación. Llamaremos vecinos a una pieza P a todas las piezas desde las cuales el jugador ubicado en P podría moverse según las reglas mencionadas anteriormente. Las reglas del juego son simples, moverse x posiciones en alguna de las 4 direcciones, donde x puede ser cualquier número entre 1 y el valor máximo en esa dirección, siempre y cuando la pieza en esa dirección sea un vecino. Consideramos un número razonable promedio de vecinos para una pieza en una matriz de N*N como N/2.

Un nuevo nodo es creado cada vez que el jugador hace un movimiento, el jugador debe hacer NxN movimientos para terminar el juego (si es un tablero lleno, si no sera T, donde T es la cantidad de piezas en el tablero, pero para simplificar las cuentas utilizaremos NxN). El camino máximo debe ser entonces de NxN, y esto nos define la altura máxima del árbol. Además sabiendo que cada pieza tiene N/2 el orden de tamaño del árbol sería de O(N/2^(n^2)), lo que realiza imposible hacer fuerza bruta para tableros mayores a 7 u 8 filas.

Motor de Busqueda

Descripción

El motor de búsqueda consiste en la expansión de nodos y la búsqueda de un camino hasta una solución. Cada Nodo posee un estado perteneciente a un problema. El problema puede definir reglas para dicho estado que generan estados subsiguientes. A este proceso se le llama explotar un nodo. El problema puede definir dado un estado si es final o no. Finalmente se puede implementar una función heurística que aproxime qué tan cerca está un estado de la solución final.

Implementacion

Para todos los métodos de búsqueda el algoritmo es el mismo. Tomar un nodo, chequear que sea solución, explotarlo, meter los nuevos nodos a un pool y repetir. La diferencia yace en la manera con la que se saca el siguiente nodo del pool y esto esta relacionado al metodo de busqueda que se esté deseando obtener.

Metodos de Busqueda

DFS

Al realizar una búsqueda DFS, es decir, en profundidad, se busca estar expandiendo los últimos nodos agregados al pool. Por lo que se necesitaba una estructura de datos que permitiera un manejo de tipo LIFO (Last In First Out), por lo que se implementó un stack. De esta manera los nodos que fueron agregados a lo ultimo seran los primeros en utilizarse. La idea detrás de esta metodología de búsqueda es poder recorrer los caminos hasta el fondo y en caso de no haber encontrado una solución, backtrackear hacia un nivel superior y continuar con otra alternativa, también en profundidad.

BFS

En este caso, se realiza una búsqueda a lo ancho, es decir que se prioriza la expansión de nodos de niveles superiores en el árbol. Este método busca encontrar el camino de longitud más corta a algún nodo solución, pero requiere una alta capacidad en espacio, ya que tiene toda la estructura del árbol en memoria. Para lograr este comportamiento se utilizó una estructura de tipo FIFO (First In First Out), para lo que se usó una queue, obteniendo el comportamiento opuesto al caso anterior.

IDDFS

La idea de iterative deepening DFS es limitar la longitud de los caminos a revisar. Se procede a revisar todos los caminos de longitud N, y una vez finalizado eso, se aumenta N y se procede a repetir lo mismo Así se evita el problema de DFS de seguir por un nivel demasiado

largo, y también limitar la cantidad de memoria utilizada en todo momento por BFS ya que solo se guarda el camino que se está recorriendo.

GREEDY

La idea es explotar los nodos cuya heurística sea menor. Greedy no tienen en cuenta el camino anterior sino solo ve el óptimo local. Se utiliza entonces una priority queue ordenada por la heurística.

ASTAR

AStar intenta explotar primero aquellos nodos que tengan menor heurística, pero siempre considerando el camino anterior. Por eso se utiliza una priority queue ordenada por la suma del costo a llegar a ese nodo más el costo de la heurística.

Heuristicas

La heurística es una métrica que aproxima, dado un estado, el costo del camino hasta la solución deseada. Dado este número, un algoritmo de búsqueda informada podría tomar decisiones en cuanto a que nodo es mas optimo para explotar.

Poda

La primera heurística que definimos fue una poda. Se podía llegar a detectar algunos estados que nunca podrán llegar a una solución. Es una heurística compuesta donde si el estado es inválido, la heurística es igual a infinito, y si no, evalúa otra heurística provista anteriormente. De esta manera se evita expandir ramas que crecen exponencialmente. La manera de determinar si un estado es inválido es ver que exista alguna pieza que ya no tiene vecinos, por lo que nunca podrá ser tocada. A pesar de que no abarca todos los casos, es un avance considerable, y continua siendo admisible ya que la distancia real para ese estado es también infinito.

Fit Factor

Necesitabamos también una heurística que apoyara a la poda. Sin embargo, al analizar el juego, nos dimos cuenta de un problema intrínseco al mismo. En cualquier momento se puede decir con exactitud la distancia al final, ya que siempre son la cantidad de fichas no tocadas. No existe una noción de caminos más alejados que otros ya que todos los caminos si llegan al final tendrán coste T, con T la cantidad de fichas. No nos sirve de nada determinar qué tan lejos está un nodo de terminar, sino más bien diferenciar cuáles son los estados más propensos a llegar al final.

Necesitamos así definir los estados con mayor cantidad de fichas los más prósperos a llegar al final, por lo analizado anteriormente en BST vs DFS. Para esto, deberíamos limitar para un estado con R piezas faltantes, su heurística sea mayor a R-1 y menor o igual a R.

Si solo se usa la distancia al final como heurística, todos los valores de coste + heurística serían iguales y se convertiría en un algoritmo BFS. Entonces habrá que modificar la heurística para que tienda a estados con mayor probabilidad de éxito.

Por esto definimos nuestras heurísticas como R - FF, donde R representa la cantidad de piezas restantes y FF es un “Fit Factor” entre 0 y 1. De esta manera, la heurística siempre será admisible y siempre considerará el Fit Factor para elegir el siguiente nodo a explotar.

A Partir de ahora haremos referencia a Fit Factor en vez de heurística. Esto es debido a que una heurística indica que tanto le falta a un estado para llegar al final. Nuestro Fit Factor en vez define que tan probable es que un estado llegue a una solución. Al momento de definir la heurística siempre será H = R - FF

Closer to end Fit Factor

Nuestro primer fit factor se basa simplemente en la elección de aquellos estados más cercanos al final, por lo que el fit factor es la cantidad de fichas tocadas. De esta manera, el algoritmo prioriza los estados con menos fichas a tocar y así poder descartarlos más rápidamente o llegar a una solución.

Less neighbours Fit Factor

El segundo fit factor constaba simplemente en la elección del estado con menor cantidad de vecinos en la pieza donde el jugador se encuentra posicionado. Una ficha con menos vecinos es más probable que sea bloqueada en un futuro (es decir que no pueda accederse de ninguna manera) si no se toma el camino por ella. Por esto, los caminos por fichas con menores vecinos son más propensos a terminar en éxito que aquellos que no.

Componiendo Fit Factors

Debido al carácter discreto y acotado de nuestras funciones Fit Factors, se puede realizar una composición. Todos aquellos estados con 3 piezas faltantes tienen el mismo valor ante el Closer to end Fit Factor, sin embargo, puede ser que alguna sea más óptima que otra debido a otro fit factor. Para esto definimos una regla de composición.

Definamos FF1 y FF2 como las 2 funciones de fit factor discretas a componer. Definimos X como la menor distancia entre dos posibles valores de FF1. Definamos M como el valor máximo de FF2. Definimos nuestro FF como $FF(E) = FF1(E) + FF2(E)/M*X$

Esta fórmula garantiza que para cualquier estado se priorice el primer fit factor y, en caso de ser iguales, ordene según el segundo fit factor. Nótese que si $FF1(E1)>FF2(E2) \Rightarrow FF(E1)>FF(E2)$ siempre.

Observaciones y Conclusiones

Es importante notar antes de comenzar a analizar los resultados de cada tipos de búsqueda, las características de nuestro problema. En caso de que el tablero tenga solución, la misma siempre consistirá en una serie de T pasos, con T siendo la cantidad de fichas. Es decir, no existirá una solución de menos de T pasos. Es por esto que BFS resultó ser una mala elección para nuestro juego. BFS expande todo el árbol, ya que debe llegar a la hoja para definir si es o no solución, haciéndolo el método más lento. Además, el orden espacial es exponencial, lo que causó graves problemas de memoria que solo lo hacían funcional en tableros chicos (ver gráfico I).

El segundo peor método es el Iterative deepening. Nunca va a haber una solucion de menos de T pasos, y el camino máximo es de T elementos, por lo que limitar la cantidad de pasos a hacer iba en contra de lo que se quería obtener (ver gráfico II).

Siguiendo esto se encuentra DFS, que es de hecho un método bastante viable. Como queremos encontrar un camino posible, seguir hasta que no se pueda más es la mejor estrategia ya que si se encuentra un solo camino que llega a T nodos, ese camino es la solución. El problema con DFS se encuentra en cuánto aumenta el tamaño del tablero. Si realiza un error al principio, DFS se dará cuenta de dicho error una vez que haya expandido todo el árbol debajo del mismo, por lo que no es buena solución para casos muy grandes.

Finalmente para arreglar este problema está la búsqueda informada. Esto nos permitirá tomar mejores decisiones en cuanto a donde explorar en el árbol. En nuestro caso, debido a que la distancia que falta es siempre dependiente del costo del nodo (ya que la distancia del camino siempre es la misma), greedy priorizará siempre los caminos más alargados, que es en parte lo que intentamos hacer con nuestra primera heurística, por lo que la diferencia entre los dos no es mucha. Procederemos a profundizar en las próximas secciones.

Por último, resta analizar el funcionamiento de las distintas heurísticas y composiciones. Como se pudo ver en los gráficos II, III y IV, el uso de una heurística resulta imperativo si se desea incrementar las dimensiones del tablero, ya que cualquier método desinformado no logra finalizar la ejecución de la búsqueda por problemas de memoria.

Aunque todas las heurísticas propuestas se muestran efectivas para tableros de dimensiones chicas, ver gráfico I, cuando se desea aumentar el tamaño, tan solo dos de ellas logran escalar (ver gráficos IV a VI). Aunque la cantidad de nodos a expandir esta determinada también por el tipo de tablero y puede no aumentar de manera lineal, se ve que son las únicas que logran resolver tableros grandes.

Se puede observar también que CFH, aunque a veces trabaje con la misma cantidad de nodos que CH, su contraparte sin filtrado, demora más tiempo (gráfico VI). Esto es debido a una mayor cantidad de operaciones intermedias que finalmente le permiten escalar aún más que CH, llegando a tableros de hasta 60x60 sin fallar por memoria (grafico VII).

Por lo que como conclusión se obtuvo que los métodos y heurísticas deseados para trabajar sobre este problema resultan ser A Estrella con heurística CFH (Compuesta con filtrado de vecinos cercanos).

Anexo

Demostración Heurísticas

Llamemos

T : La cantidad de fichas

E_R : El estado con R fichas sin tocar

$C(N_e)$: El costo de llegar al nodo N_e

$P(N_e) = C(N_e) + H(E)$ el ordenamiento en la cola

$$P(N_{E_R}) < P(N_{E_{R+1}})$$

$$C(N_{E_R}) + H(E_R) < C(N_{E_{R+1}}) + H(E_{R+1})$$

$$T - R + H(E_R) < T - R - 1 + H(E_{R+1})$$

$$H(E_R) < H(E_{R+1}) - 1$$

$$H(E_R) < H(E_{R+1}) - 1 \quad \rightarrow \quad R - 1 < H(E_R) \leq R$$

$Caso\ Base) R = 0$ $\quad 0 \leq H(E_0)$ *ya que debe ser positivo*

$H(E_0) \leq 0$ *ya que es admisible*

$-1 < H(E_0) \leq 0$

Paso Inductivo) $\quad R - 2 < H(E_{R-1}) < H(E_R) - 1 \quad \rightarrow \quad R - 1 < H(E_R)$

$y \quad H(E_R) \leq R \quad$ *por ser admisible*

$R - 1 < H(E_R) \leq R$

Sección Gráficos

Los métodos A-XXX y G-XXX representan los metodos de busqueda A Estrella y Greedy respectivamente, y el conjunto de letras que los siguen muestran la heurística, o composición de heurísticas que utilizaron

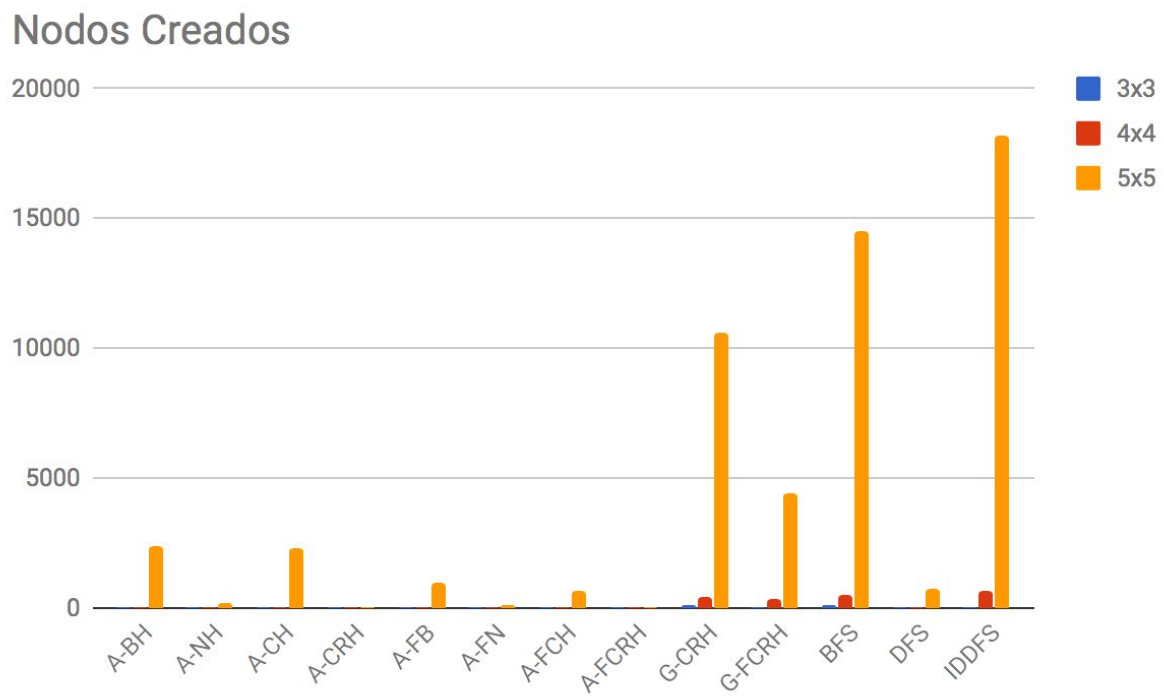


Gráfico I)Cantidad de Nodos creados por los distintos métodos para tableros de 3x3, 4x4 y 5x5.
Se observa una notable diferencia al utilizar A-Estrella con heurísticas que proveen información sobre el problema

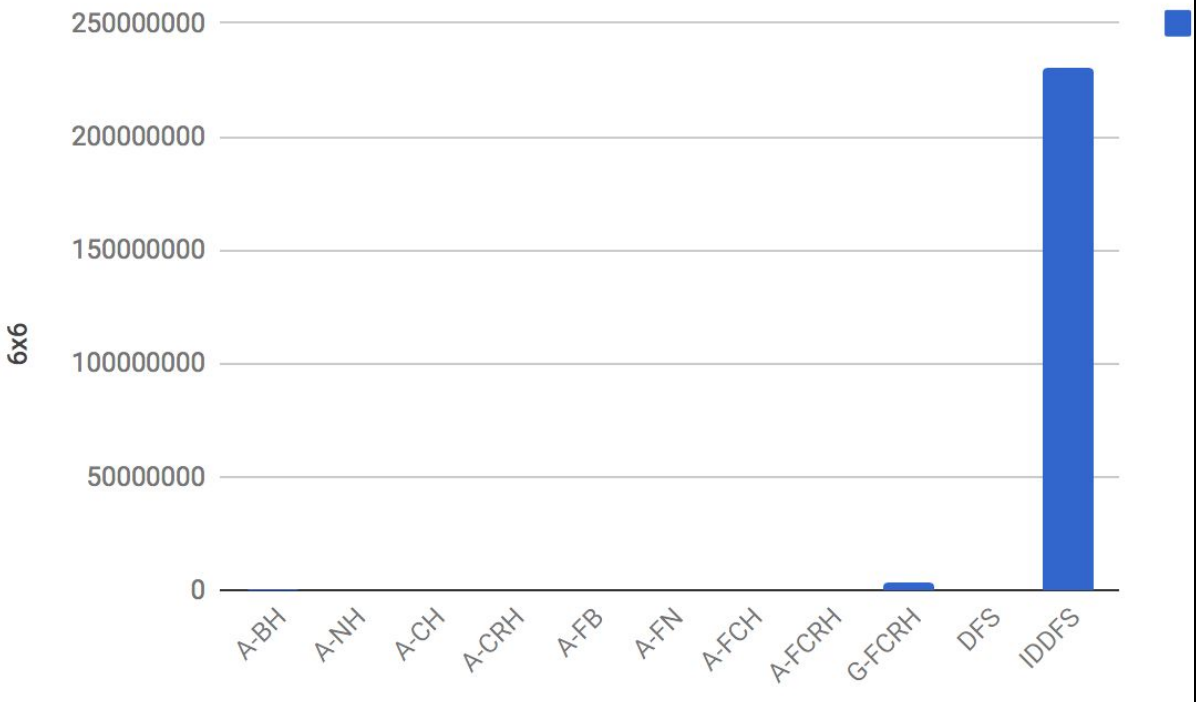


Gráfico II) Nodos creados para tablero de 6x6

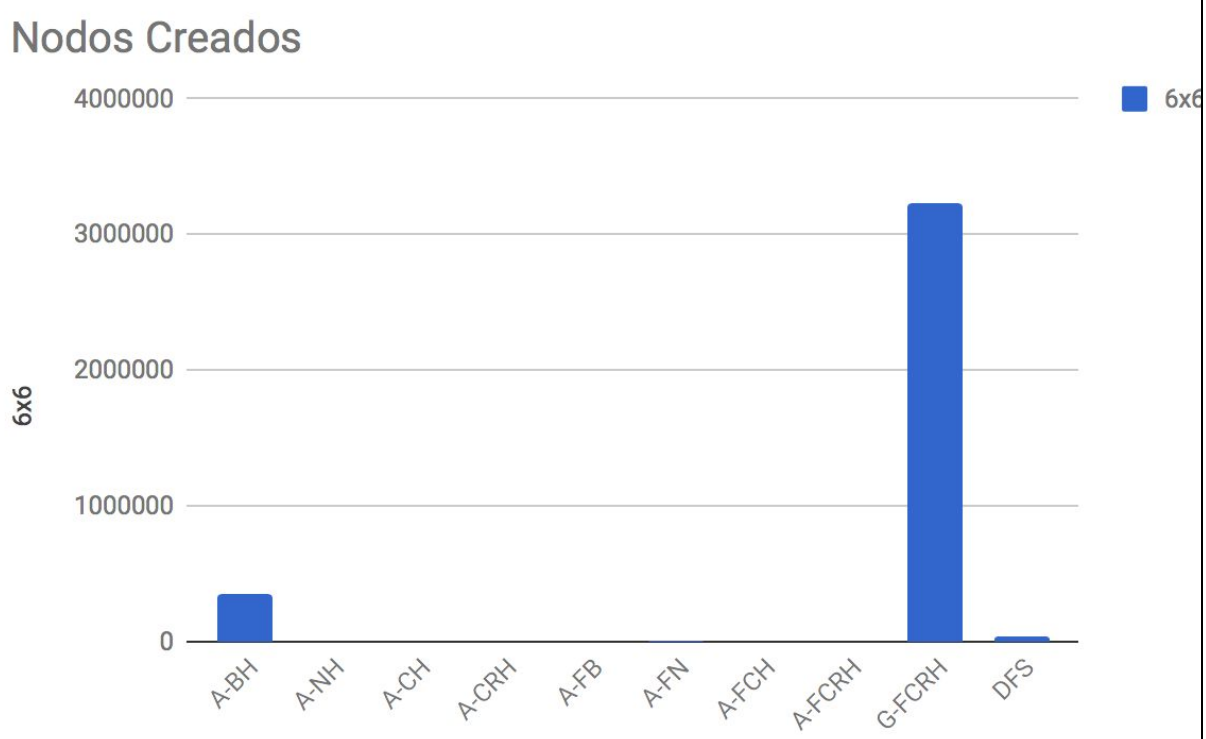


Gráfico III) Nodos creados para tablero de 6x6, removiendo el método de profundidad iterativo

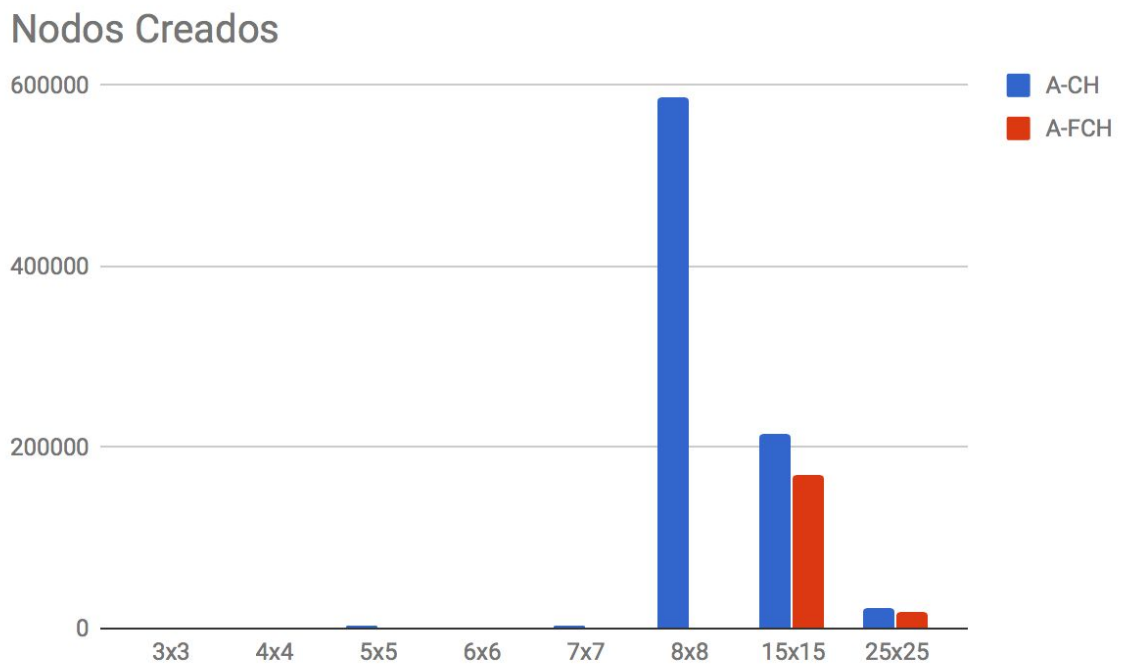


Gráfico IV) Nodos creados por A-Estrella con las dos mejores heurísticas que logran escalar en tamaños de tableros

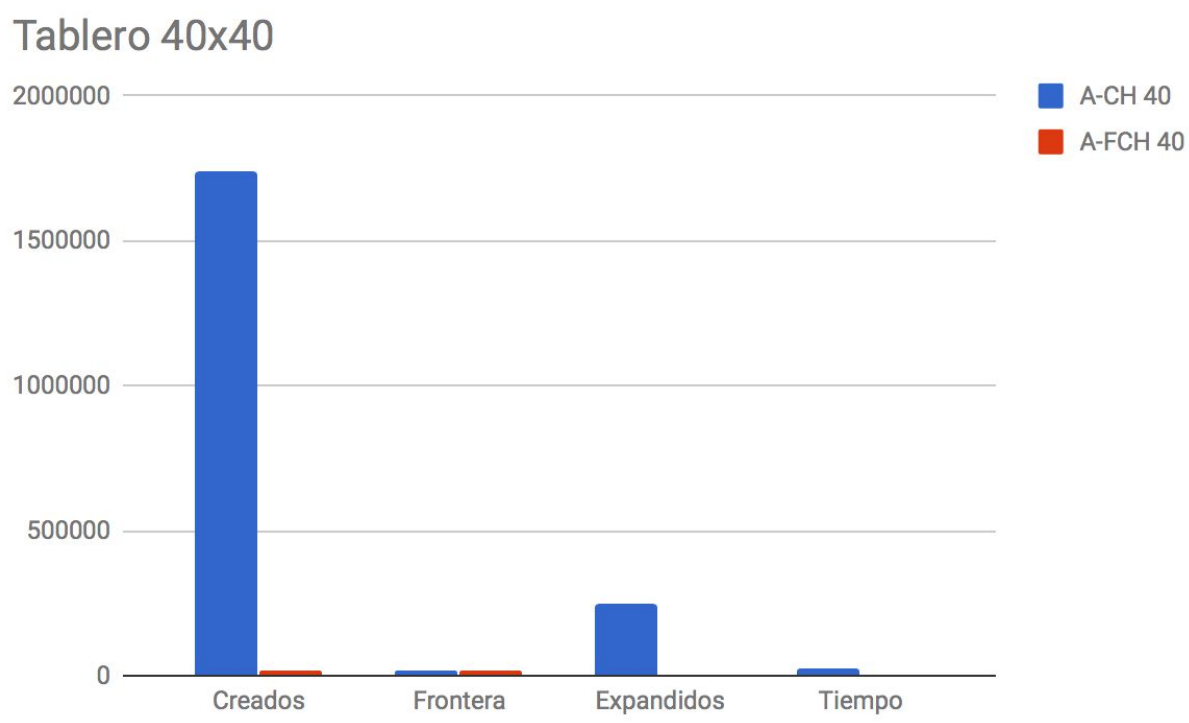


Gráfico V) Resultados para tablero de 40x40 con las dos heurísticas que lograron escalar.

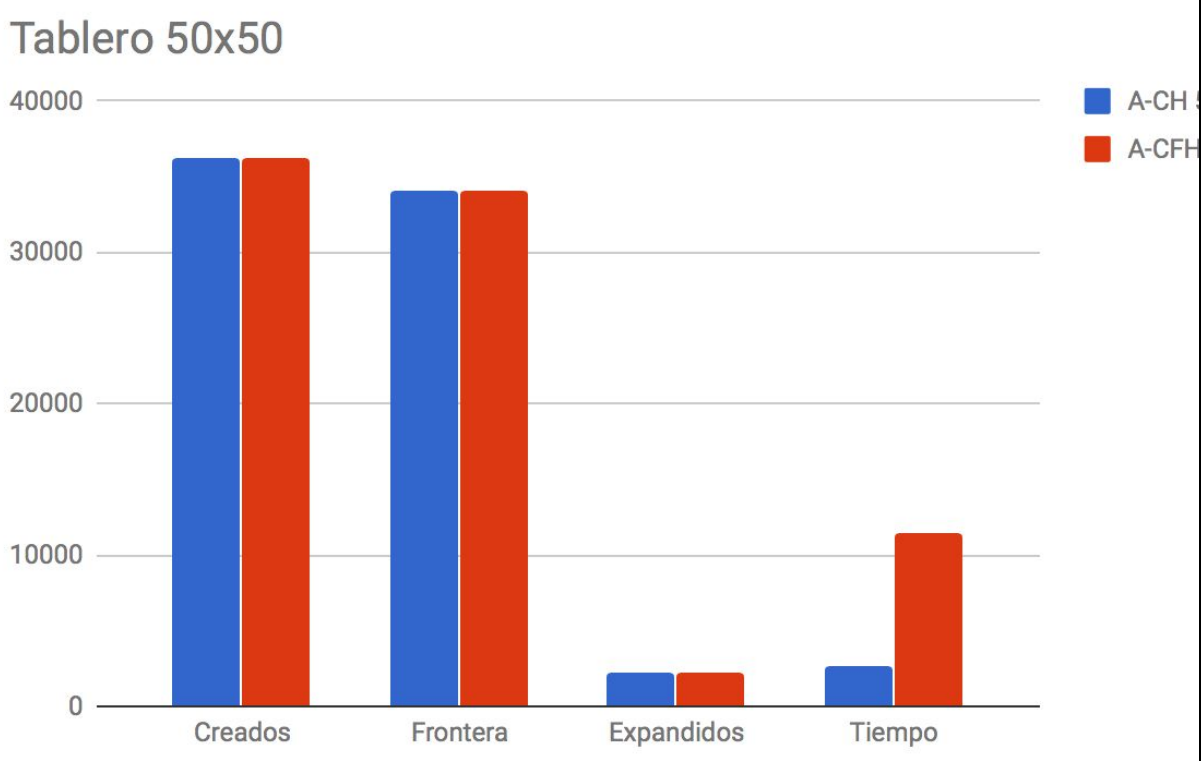


Gráfico VI) Resultados para tablero de 50x50. Nótese que a pesar de trabajar con la misma cantidad de nodos, la heurística filtrada demora un tiempo mayor

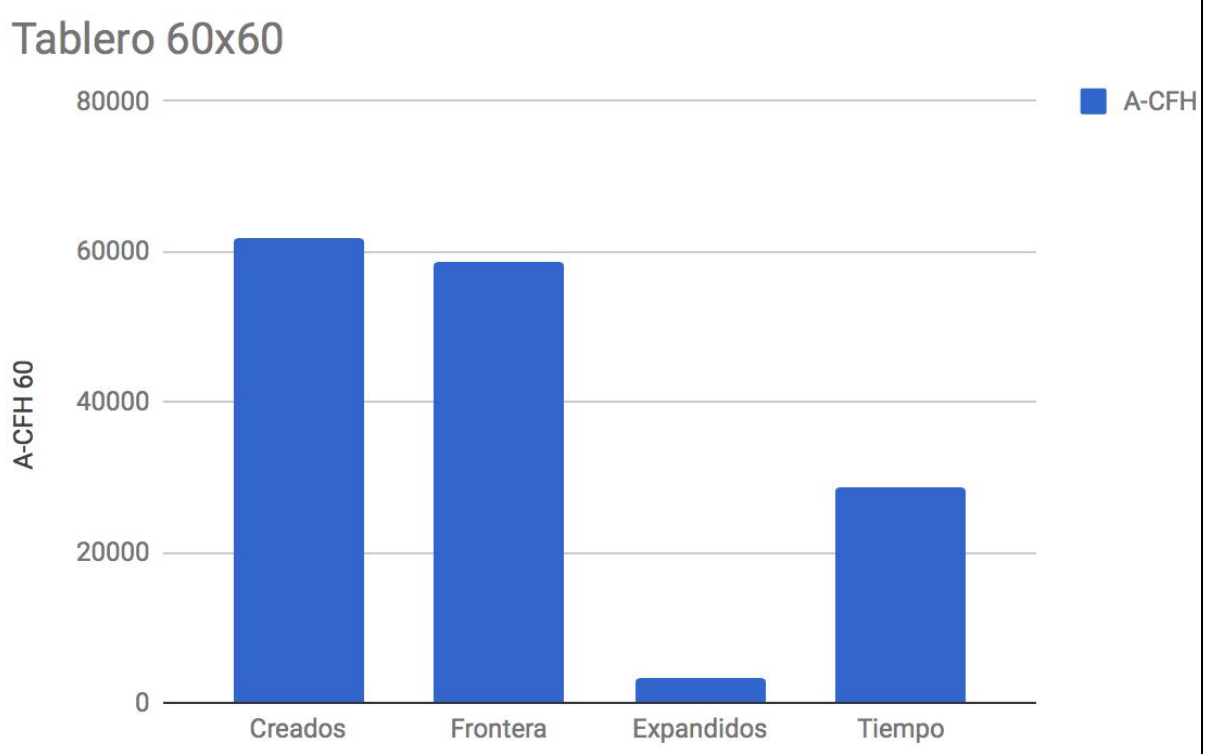


Gráfico VII) Resultados para tablero de 60x60, tan solo la heuristica filtrada logra resolverlo.